Computer Science Department

TECHNICAL REPORT

On the Detection of Robust Curves

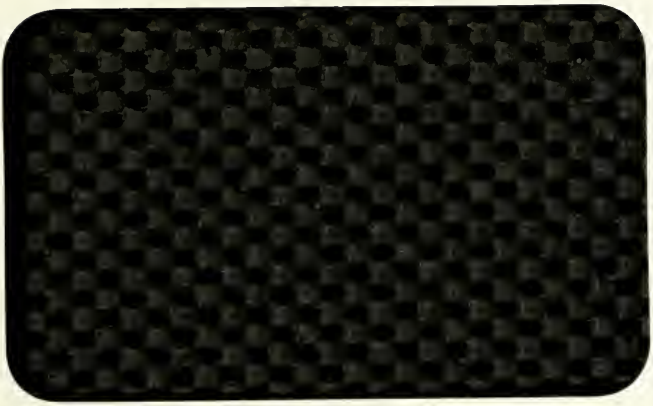
Richard Cole
Uzi Vishkin

Technical Report 557

April 1991

NEW YORK UNIVERSITY

Department of Computer Science
Courant Institute of Mathematical Sciences
251 MERCER STREET, NEW YORK, N.Y. 10012



On the Detection of Robust Curves

Richard Cole
Uzi Vishkin

Technical Report 557

April 1991

On the Detection of Robust Curves

Richard Cole*
Courant Institute
New York University

Uzi Vishkin†
University of Maryland &
Tel Aviv University

April 22, 1991

Abstract

Given m points in the plane and a threshold t, a curve is defined to be robust if at least t points lie on it. Efficient algorithms for detecting robust curves are given; the key contribution is to use randomized sampling. In addition, an approximate version of the problem is introduced. A geometric solution to this problem is given; it too can be enhanced by randomization.

These algorithms are readily generalized to solve the problem of robust curve detection in a scene of curve fragments: given a set of curve segments, a curve σ is defined to be robust if curve segments of total length at least l lie on σ. Again, both an exact and an approximate version of the problem are considered.

The problems and solutions are closely related to the well-investigated Hough Transform technique.

1 Introduction

A recent survey paper by Illingworth and Kittler [IK88] refers to the Hough Transform, HT, as "... a technique of almost unique promise for shape and motion analysis in images containing noisy, missing, and extraneous data but its adoption has been slow due to its computational and storage complexity ...", and cites 144 papers written on it by 1988.

On the skeptical side, a few papers (e.g., [Br83] and [GH90]) offer criticism of the HT technique. Most criticism is directed towards the sensitivity of the HT and suggests not to use it blithely.

While the present paper is inspired by HT, it applies a notion of *robust curves* to overcome the sensitivity related criticism, while maintaining the power of HT. We hope that our algorithms will contribute ideas to designers of software systems for scene analysis.

*Partially supported by NSF grants CCR-8906949 and 8902221.
†Partially supported by NSF grant CCR-8906949.

Our main concern in this presentation is to address computational (and storage) efficiency issues.

Given m points in the plane and a threshold t, a curve is defined to be robust if at least t points lie on it. Alternatively, given m line segments in the plane and a length threshold l, a curve is defined to be robust if line segments of total length at least l lie on it. We give efficient algorithms for detecting robust curves under both definitions (the "RSL problem"). Section 3 deals with problems where the points (or line segments) lie exactly on a curve. The main contribution is to apply randomization. In Section 4, we introduce a similar problem (the "ϵ-RSL problem"), where points (or line segments) are counted even if they only lie near a curve. A geometric solution which can be enhanced by randomization is given.

A powerful algorithmic methodology which is used extensively in the present paper is randomization. We use the concept of randomized algorithms in the same spirit as advocated by Rabin [Ra76]: The algorithm "flips a coin" in order to determine its next move. We then analyze the time and space requirements of the algorithm. Upper bounds for its expected running time, or for its running time with high probability, are given. To avoid misunderstanding, we emphasize that our analysis does not make any assumptions regarding a probabilistic distribution of the input; all our results hold for any input. Previous use of randomization in the context of HT was proposed in [FB81,FF81,XOK90]; no analysis was given in these papers. Our method of randomization is different. Another recent paper, by Bergen and Shvaytser [BS90], also advocates the use of randomization for achieving faster implementations of the Hough transform. One major difference between the paper by Bergen and Shvaytser and ours lies in the algorithms; a substantial part of our paper is concerned with developing new algorithms, whereas Bergen and Shvaytser are concerned with more efficient implementations of the standard approaches. Our algorithms are more efficient.

A novel element in our treatment is that its primary concern is with algorithmic efficiency. There is also a contribution in introducing a (provably) precise approximation in the algorithm for the ϵ-RSL problem. While the real numbers are presumably represented in finite precision and hence are discretized, our approach has the advantage that our algorithms do not impose a further coarser discretization. "Traditional" image processing algorithms (e.g., [BS90]) include explicit steps that chop off all but a certain number of the most significant bits of some real values (either inputs or values computed from the inputs). This leads to (unquantified) approximate results. Actually, it is not clear whether these approximate results can be quantified, by contrast with our work. (It should be noted that our algorithms do not concern themselves with the question of coping with the finite precision of computer arithmetic; it is assumed that all arithmetic computations with reals are of sufficient precision and take constant time.)

We draw attention to two aspects of the present paper:

1. Rather than seeking to find the best unique solution to a problem, as in much of the published work on algorithms, our output definition seeks all solutions that meet the

problem specification; we are thereby reducing the search space of possible solutions for further processing, an approach common in the computer vision and artificial intelligence communities.

2. We believe that the approximate problem, that is, *the recognition of curves in an image comprising points, where the image is noisy*, is an important practical problem. In defining it, our treatment is less conservative than the "standard image processing approach:" Observing that the Hough transform is a means rather than a goal, we avoid trying directly to give an efficient implementation of the Hough transform; instead, we focus on the problem being solved via the Hough transform and solve that problem efficiently.

We describe sequential and parallel implementations of our algorithms. We note that the parallel algorithm for the ϵ-RSL problem is quite different to the serial algorithm, and actually provides an alternative serial algorithm. The parallel algorithms are for the CREW PRAM model. This model comprises a set of P processors together with a shared memory. Each processor is able to access any memory location in constant time. However, while concurrent reads are allowed, concurrent writes are forbidden. It is traditional to state complexity results as a (P, T) pair, where P denotes the number of processors used and T the parallel time. We prefer to use the pair (W, T), where T denotes the parallel time, as before, and W denotes the work or the number of operations performed; $W = P \cdot T$. An algorithm that has complexity (W, T) can be implemented to run on P processors in parallel time $O(\frac{W}{P} + T)$; the advantage of the (W, T) notation is that it shows at a glance the efficiency of the algorithm (expressed by the W term) and the fastest parallel time that can be achieved efficiently (or equivalently the maximum amount of parallelism that can be efficiently used, namely $\frac{W}{T}$ processors).

Our style of presentation emphasizes ideas whose effectiveness can be proven. Other ideas, which seem useful but whose effectiveness is not proven, will typically be deferred to comments. While the algorithms we develop are all new, we will note wherever components can be traced to algorithmic paradigms.

2 Preliminaries

We start by reviewing the concept of the Hough Transform. See also [BB82] for details and justifications. Let $M = \{(x_1, y_1), (x_2, y_2), ..(x_m, y_m)\}$ be a set of m points in the plane. Observe that any straight line in the plane can be described as a pair (r, θ) defined as follows. For the purposes of our definition and in order to distinguish between lines crossing the x-axis to the left and the right of the origin, we define the lines to be oriented so that the origin is to their right. A line through the origin is oriented in the increasing y direction. r gives the normal distance of the line from the origin. The parameter θ gives the angle from the oriented x-axis to the oriented line, measured anticlockwise. See Figure 1. θ can take any value in the range $[0, 2\pi]$ and r is a real number.

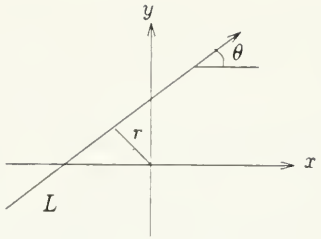


Figure 1: Definition of r and θ for line L

Comment. The above definition is not continuous at $r = 0$. In Section 4, we will want a continuous mapping. To achieve continuity, we map each line to two points, namely (r, θ) as defined above and $(-r, \theta + \pi)$ (where all angle computations are mod 2π). Next, we explain why continuity is achieved. Consider a line L with a fixed orientation that is advanced from the left side of the origin to the right side. The two-point mapping gives the pair of points (r, θ) and $(-r, \theta + \pi)$, where $r \geq 0$. When the line crosses the origin, this pair switches to $(r, \theta + \pi)$ and $(-r, \theta)$, respectively, with $r \geq 0$ still.

All straight lines that contain the point (x, y) must satisfy the following equation

$$r = x \sin\theta - y \cos\theta \tag{1}$$

The plane with respect to parameters θ and r is called the *dual* plane. Applying equation 1 separately to each point in M transforms M into a set of m sinusoidal curves in the dual plane. Consider two points in M. They define two curves in the dual plane. The points of the dual plane in which these two curves intersect gives the unique straight line in the (primal) plane that includes the two points of M. In fact, it suffices to restrict attention to the θ interval $[0, \pi]$; then two of these sinusoidal curves, if not identical, intersect in a single point.

Applications of HT in image processing often look for intersection points among pairs of curves in the dual plane. A point in which several curves intersect in the dual plane is considered meaningful since it implies that the corresponding straight line in the plane includes several points of M. Also, clusters of intersection points in the dual plane are considered meaningful, since they seem to "vote" for the same "object." m points will induce $\Theta(m^2)$ intersection points in the dual plane.

In the present paper we avoid computing all intersection points explicitly. This will enable our algorithms to perform $\ll m^2$ operations (where $\ll$ means smaller in order of magnitude) for the problems we define. This is the main novel feature of our results. The problems we define emphasize finding straight lines (or other curves) that many points in M vote for. Other aspects of HT, including the fact that this is a transform, are less relevant for our problems.

Two probabilistic propositions

The following two propositions will be helpful. Proposition 1 is a variant of Chernoff's bounds, which is due to [AV79].

Proposition 1 *For all n, p, β with $0 \leq p \leq 1$, $0 \leq \beta \leq 1$,*

$$\sum_{k=0}^{\lfloor (1-\beta)np \rfloor} \binom{n}{k} p^k (1-p)^{n-k} \leq \exp(-\beta^2 np/2) \tag{2}$$

$$\sum_{k=\lceil (1+\beta)np \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k} \leq \exp(-\beta^2 np/3) \tag{3}$$

Proposition 2 will be used for lower bounding the joint probability of two events which are not necessarily independent.

Proposition 2 *Let E_1 and E_2 be any two events, and suppose that $Prob(E_1) \geq 1 - \delta$ and $Prob(E_2) \geq 1 - \delta$. Then $Prob(E_1 \bigcap E_2) \geq 1 - 2\delta$.*

3 Robust Straight Lines

Let M be a set of m points in the plane and let an integer t, $1 \leq t \leq m$, be called a *threshold*. A straight line that t, or more, points lie on is called *robust*. Likewise, let S be a set of s line segments in the plane and let $l > 0$, a real number, be called a *length threshold*. A straight line that contains segments of total length at least l is called *robust*. It will always be clear from the context in which sense the term robust is being used. In this section we study the following two problems.

The Point Robust Straight Lines (point-RSL) Detection Problem.
Input. Set M of points and threshold t.
Problem. Find all straight lines that are robust.

The Segment Robust Straight Lines (segment-RSL) Detection Problem.
Input. Set M of points and threshold l.
Problem. Find all straight lines that are robust.

The RSL problems are simpler than the problems of Section 4 since for each line, only objects that lie exactly on it are taken into account.

We begin by considering the point-RSL problem.

We say that two points in the plane *define* the line they lie on. Let U be a subset of M. The *lines of* U are those lines defined by (at least) two points in U. It is convenient to solve the following more general problem.

The Subset-RSL Problem. Find all the straight lines of U that are robust.

Overview of the algorithms for the point-RSL problem. Algorithm 1, below, provides a framework for the algorithms in this section. The implementation of the algorithm

is described in some detail. However, the main concern in this section is not Algorithm 1, but rather how to speed it up, which is done in Algorithms 2 and 3. The main contributions are: (1) Using random sampling to select the subset U. (2) Using only a subset of M to determine which lines of U are robust. This subset too is picked by random sampling.

3.1 Algorithm 1 - the Framework Algorithm

Algorithm 1 is for the subset-RSL problem. The point-RSL problem itself is solved by choosing $U = M$.

Algorithm 1

Step 1: Find the lines of set U and store them efficiently.

Implementation of Step 1. The lines are stored by means of hashing; thus, given any line, we can check whether it is also a line of U in $O(1)$ time. For the parallel (and perhaps even the serial) implementation we suggest using the randomized parallel algorithm of Matias and Vishkin [MV90]; it follows the non-constructive scheme of Fredman, Komlos and Szemeredi [FKS84]. Storing all $\binom{|U|}{2}$ lines takes $O(\log|U|)$ time using $O(|U|^2)$ operations and $O(|U|^2)$ space. The time and operation counts are expected, while the space is in the worst case. Finding a line will take $O(1)$ operations, in the worst case. An alternative serial implementation, with the same serial complexity results, may rely on the hashing algorithm of [DKM+88]. This hashing algorithm also follows [FKS84].

Step 2.1: For each line of U, determine how many points in U lie on it.

This is done as follows. For each pair of points in U increment a counter associated with the line they define. A line of U that x points of U lie on it is counted $\binom{x}{2}$ times. A solution to a quadratic equation gives x. Serially, this takes $O(|U|^2)$ operations.

Step 2.2: For each line of U, determine how many points in M lie on it.

This is done as follows. For each point x in U and each point y in $M - U$, determine whether the straight line they define is also a line of U, and if so increment a counter associated with this line. Consider a line such that $h \geq 2$ points of U and i points of $M - U$ lie on it. Following the computation, the counter for this line will store the number $h \cdot i$. So it remains to divide this number by h. Serially, this (and the whole of Step 2) will take $O(|U| \cdot |M|)$ operations.

Parallel Implementation of Step 2. In parallel, Step 2 can be performed in expected $O(\log|M|)$ time and expected $O(|U| \cdot |M|)$ operations. We discuss Step 2.2. The parallel implementation of Step 2.1 is similar. The pairs of points x and y are processed in parallel; the result of a computation on x and y is either the line L defined by U on which x and y lie or the result "no line." These $|U|(|M| - |U|)$ results are sorted using the randomized integer sorting algorithm of Rajasekaran and Reif [RR89]. Some care is needed since this sorting algorithm tolerates inputs only from a restricted domain; e.g., the domain of

integers $[1 \cdots O(|U| \cdot |M|)]$. This restriction is met as follows. A property of the hashing algorithm of Matias-Vishkin (and, actually of the Fredman-Komlos-Szemeredi algorithm) is that each straight line in U will be labeled by a separate integer between 1 and $5|U|^2$. So, it suffices to apply the Rajasekaran-Reif sorting algorithm to these labels. Finally, the number of occurrences of each line is counted using the prefix sum algorithm of Ladner and Fischer [LF80]. These algorithms both perform a linear ($O(|U| \cdot |M|)$) number of operations and use $O(\log |M|)$ time.

We have shown:

Theorem 1 *Suppose that u lines of U are robust. Then the above algorithm will find all of them in expected $O(|U| \cdot |M|)$ time using $O(|U|^2)$ space. In addition, there is a parallel implementation of the algorithm which runs in expected $O(\log |M|)$ time and expected $O(|U| \cdot |M|)$ operations, using $O(|U|^2)$ space.*

Comment. Hashing in combination with floating point arithmetic is problematic. So we imagine these algorithms being used for integer or fixed precision coordinates to which integer arithmetic can be applied.

3.2 The Improved Algorithms

Algorithm 2

In order to achieve greater efficiency, rather than choose $U = M$, U is selected by random sampling in a preprocessing Step 0. So let p be a probability parameter, $p = (2 \ln m + 2)/t$. (implicitly, we are assuming $t \geq 2 \ln m + 2$).

Step 0: Select each point in M with (independent) probability p into a set U. Note that U has expected size pm (which is at least $(2 \ln m + 2)$.

Otherwise, Algorithm 2 proceeds as Algorithm 1.

The next lemma and corollary follow from Propositions 1 and 2, respectively.

Lemma 3 *Let A denote a robust line. Then, the probability that at least two points that lie on A were selected for U (formally, $|A \cap U| \geq 2$) is at least $1 - 1/m$.*

Proof. We apply Equation 2 (from Proposition 1) with $\beta = (tp - 1)/tp$. Since $|A| \geq t$, $Prob(|A \cap U| \leq 1)$ is at most

$$exp(-\frac{(tp-1)^2}{(tp)^2}\frac{tp}{2}) \leq exp(-\frac{tp-2}{2}) \leq exp(-\ln m) \leq 1/m$$

The lemma follows. •

Corollary 4 *Suppose that there are r robust lines. Then, all of them will be defined by U with probability at least $1 - r/m$.*

Theorem 2 *Suppose that r lines of M are robust. Then:*

1. *Algorithm 2 will find all of them with probability at least $1 - r/m$.*
2. *The algorithm runs in expected $O(|U| \cdot |M|)$ time using $O(|U|^2)$ space.*
3. *There is a parallel implementation of the algorithm that runs in expected $O(\log |M|)$ time and expected $O(|U| \cdot |M|)$ operations, using $O(|U|^2)$ space.*

Comment. Larger values of p result in smaller probabilities of failure at the cost of a larger running time. For note that the expected size of U is a linear function of p.

Algorithm 3

Steps 0, 1 and 2.1 are as in Algorithm 2.

We apply the sampling process in Step 2.2, also. However, there is no longer a guarantee of clearcut results for lines which have a number of points close to the threshold. Instead, with high probability, the algorithm reports all lines on which at least t points lie; it will also report some lines on which close to t points lie. This is made precise below. First, we describe the modified algorithm.

A more efficient implementation of Step 2.2. The idea is to sample a subset M' of M uniformly at random and then estimate which lines of U are robust based on how many points of M' lie on them.

Step 2.2.1:* Select each point in M with (independent) probability q into a set M'. Note that the expected size of M' is $q|M|$.

Step 2.2.2:* For each point x in U and each point y in M', find whether the straight line they define is also a line of U, and if so increment a counter associated with this line. Serially, this will take $O(|U| \cdot |M'|)$ operations. In parallel, this can be done in expected $O(\log |M'| + \log |U|)$ time and expected $O(|U| \cdot |M'|)$ operations, using the method of Step 2.2 above.

¿From Proposition 1, we deduce:

Lemma 5 *Let A denote a line. Suppose some s points in M' lie on A. Let β and q be constants with $0 \leq \beta \leq 1$ and $0 \leq q \leq 1$.*

1. *If A is robust, then $s \geq (1 - \beta)tq$ with probability at least $1 - \exp(-\beta^2 tq/2)$.*
2. *If $s \geq (1 + \beta)tq$, then A is robust with probability at least $1 - \exp(-\beta^2 tq/3)$.*
3. *If $s \geq (1 - \beta)tq$, then at least $(1 - \frac{1-\beta}{1+\beta})t$ points in M lie on A with probability at least $1 - \exp(-\beta^2 \frac{1-\beta}{1+\beta} tq/3)$.*

This lemma suggests using the threshold test of being incident on at least $(1 - \beta)tq$ points in M' to identify robust lines. Clause (1) shows that a certificate of certainty can be attached to lines that exceed a threshold of $(1 + \beta)tq$ points and clause (3) indicates which additional non-robust lines may be reported.

In particular, choosing $M' = U$ (and so $p = q$) and using Proposition 2 we obtain:

Corollary 6 *Suppose that r lines of M are robust. Then:*

1. *The above algorithm will find any one line with probability at least $1 - \exp(-\beta^2 tp/2)$.*
2. *It reports all r robust lines with probability at least $1 - r\exp(-\beta^2 tp/2)$.*
3. *It also reports any given line with fewer than $(1 - \frac{1-\beta}{1+\beta})t$ points with probability at most $\exp(-\beta^2 \frac{1-\beta}{1+\beta} tp/3)$.*
4. *In addition, it reports no lines with fewer than $(1 - \frac{1-\beta}{1+\beta})t$ points with probability at least $1 - m^2 \exp(-\beta^2 \frac{1-\beta}{1+\beta} tp/3)$.*
5. *The algorithm runs in $O(|U|^2 + |M|)$ time using $O(|U|^2)$ space.*
6. *There is a parallel implementation of the algorithm that runs in expected $O(\log |M|)$ time and expected $O(|U|^2 + |M|)$ operations, using $O(|U|^2)$ space.*

Remark 1. To give a more intuitive feeling for the meaning of Corollary 6, we substitute some example numbers for the parameters. Suppose $m = 10^6$ and $t = 10^4$. Choosing $p = \frac{1}{50}$ and $\beta = \frac{1}{3}$ yields $\exp(-\beta^2 \frac{1-\beta}{1+\beta} tp/3) \leq \frac{1}{20}$. This means that any given robust line is missed with less than 5% probability, and any line which is far from robust (on which fewer than $(1 - \frac{1-\beta}{1+\beta})t = \frac{t}{2}$ points lie) is reported with at most 5% probability. However, as m^2 is large, and likewise if r is large, the probability of some error need not be small. Nonetheless, as we discuss is Remark 4, below, the results provided by the algorithm may still be useful.

Additionally, it is our belief that bounds of Corollary 6 substantially overstate the probability of error. Several simulations were run to confirm this. The following values were experimented with: $m = 2000$, $t = 250$ and p in the range between 0.025 and 0.15. Robust lines were missed in only one out of forty experiments. Also, for the smaller values of p in this range, some non-robust lines were (wrongly) reported as robust. However, in general robust lines included more sample points than non-robust lines.

Remark 2. Another approach is to filter the lines reported using this algorithm, by checking each line reported against the full point set M. This is efficient only if there are relatively few lines on which close to t points lie. Indeed, we could have a series of stages in which larger and larger sample sets M' are used to check the remaining doubtful lines; clauses (1) and (2) of the above lemma can be used to exclude lines for which the result (robust/not robust) is known with high probability. (We note that in Algorithm 1 we achieved a runtime of $O(|U| \cdot |M|)$, because each point of M could lie on at most one line defined by each point of U; this yields a saving of a multiplicative factor of $|U|$ over the runtime of the naive algorithm, which compares each point of M against each of the $|U|^2$ lines. We do not achieve similar gains over the naive algorithm in verifying the remaining lines during the filtering process; the verification algorithm, in each phase, runs in time equal to the minimum of $O(|U| \cdot |M'|)$ and $O(u' \cdot |M'|)$, where u' is the number of lines at hand and M' is the point set being considered.) In a parallel implementation the following bounds can be achieved: the minimum of expected $O(\log |M'|)$ time and expected $O(|U| \cdot |M'|)$ operations (using the method described earlier) and of $O(\log |M'|)$

time and $O(u' \cdot |M'|)$ operations (using the obvious naive algorithm to compare each line with each point of M' and the prefix sum algorithm of Ladner and Fischer [LF80] to collate the results).

Remark 3. Another approach to the filtering problem is based on point location algorithms [DL76,ST86]. Again, let u' be the number of lines reported as having close to t points lying on them. The following procedure is used. Build a point location data structure for these u' lines. Such a data structure can support the following type of query: given a query point it reports the line(s) on which the query point lies, or which lines lie immediately above and below the query point; further, answering a query takes $O(\log u')$ time. The data structure of [ST86] requires $O((u')^2)$ space and can be built in $O((u')^2 \log u')$ time, while that of [DL76] requires $O((u')^3)$ space and can be built in $O((u')^3 \log u')$ time (however, the latter data structure is particularly simple). Given this data structure, for each point of M in turn, determine on which of the u' lines, if any, it lies. For each of the u' lines maintain counters of how many points of M lie on them. The querying takes $O(|M| \log u' + (u')^2)$ time overall. For each query takes time $O(\log u')$ plus $O($the number of lines reported). But, as all the points of M are distinct, the total number of multiple lines reported in queries can be at most $O((u')^2)$, since each intersection point of the u' lines is reported at most once and the total number of such points, even counted according to their multiplicity (i.e., the number of lines meeting in the intersection point) is $O((u')^2)$. So this approach yields a verification algorithm with running time $O(|M| \log u' + (u')^2 \log u')$, which is efficient if u' is relatively small. Next, a parallel implementation is described. It uses the planar point location data structure of Atallah et al. [ACG89], which can be constructed in $O(\log u')$ time and $O((u')^2 \log u')$ operations and answers queries in $O(\log u')$ time. The query is answered in two phases: in the first phase, the number, n_q, of lines on which the query point, q, lies is determined; then, each point q is allocated n_q processors, each processor being responsible for reporting a separate line on which q lies. (This process is not described explicitly in [ACG89] but the algorithm given there is readily modified to support this form of response.) The set of query point/line incidences is now sorted (eg. by Cole's algorithm [Col88]), using the line label as the key; next, using a prefix sum algorithm ([LF80]), for each line, the number of query points on the line is determined. Overall, this requires $O(\log u')$ time and $O(|M'| \log u' + (u')^2 \log u')$ operations. It should be noted that the parallel planar point location algorithm of Atallah et al. is quite elaborate; a simpler algorithm, which achieves somewhat less parallelism, is given in [AG86].

Remark 4. The performance of our algorithms degrades as r, the number of robust lines, increases. We feel that our algorithms can be useful even if they find only a majority of the lines that exist in some scene. For we intend our algorithms to provide ideas which will be incorporated within software systems for scene analysis. Scene analysis can be viewed as a search problem over a huge domain. Even an initial detection of a few robust lines can reduce drastically the complexity of the remaining problem. The remaining problem might be dealt with in an interactive fashion with human intervention, or automatically by the software system itself. We also note that the point-RSL problem takes no advantage

of the fact that an image is being processed; presumably there is semantic information to be exploited. This however, is outside the problem domain we are considering. But it indicates that the role of the point-RSL algorithm should be to find a substantial portion of the robust lines in order that further processing can proceed effectively.

3.3 The Segment-RSL Problem

This problem is solved using algorithm 1 with two modifications. First, the set S of line segments replaces the lines of point set U. Second, in Step 2, for each line, A, containing a line segment of S, a length counter is maintained: it accumulates the total length of all line segments in S lying on A. Clearly, the algorithm takes $O(|S|)$ time.

Sampling is not used here, but randomization is still present through the use of hash functions.

In a parallel implementation it is not clear how to carry out Step 2 in $O(|S|)$ operations. So we give an implementation that performs $O(|S| \log |S|)$ operations in $O(\log |S|)$ time. This implementation does not use any randomization. Clearly, an analogous sequential algorithm could be given.

Step 1. Sort the line segments of S by slope and by their intersection with the x-axis (this ensures that segments lying on a common line are adjacent in the sorted order).

Step 2. Determine the length of the segments on each common line by means of a prefix sum algorithm.

Using Cole's sorting algorithm [Col88] for Step 1 and the Ladner-Fisher prefix sums algorithm [LF80] for Step 2 achieves the claimed complexity.

We have shown:

Theorem 3 *There is a sequential algorithm for the segment-RSL problem which runs in expected $O(|S|)$ time. Likewise there is a parallel algorithm for this problem which performs $O(|S| \log |S|)$ operations in $O(\log |S|)$ time.*

4 ϵ-Robust Straight Lines

We proceed to the two other main problems considered in the present paper.

Let M be a set of m points in the plane, as before. Let A be *any* straight line in the plane. A point in the plane is ϵ-near to A if its normal or Euclidean distance from A is at most ϵ, for some $\epsilon > 0$. A is called *ϵ-robust* if at least t points, for some $t > 0$, are ϵ-near to A.

The Point ϵ-Robust Straight Lines (Point ϵ-RSL) Detection Problem.
Input. Set M of points, threshold t and $\epsilon > 0$.
Problem. Find all straight lines that are ϵ-robust.

Likewise, let S be a set of s line segments in the plane. Let A be *any* straight line in the plane. A line segment σ is ϵ-near to A if every point on σ is ϵ-near to A. A is called *ϵ-robust* if segments of total length at least l are ϵ-near to A. Possible alternate definitions of the notion of ϵ-robust for line segments will be discussed later. Again, it will always be clear from the context in which sense the term ϵ-robust is being used.

The Segment ϵ-Robust Straight Lines (Segment ϵ-RSL) Detection problem.
Input. Set S of line segments, threshold l and $\epsilon > 0$.
Problem. Find all straight lines that are ϵ-robust.

Where it is clear from the context that the ϵ-RSL problem is being discussed, we omit the prefix ϵ. That is, we simply say that a point or line segment lies on a straight line and that a line is robust.

The main contribution of this section is a so-called geometric algorithm for the point-RSL problem; it reports all the (regions of) robust lines in time $O(m^2)$. Later, we discuss how this algorithm can be speeded up using random sampling (at some cost in certainty).

4.1 The Geometric Algorithm for the Point-RSL problem

In the description of the algorithm that follows, the region containing the points ϵ-near to a line A is called the radius ϵ *cylinder* with *axis* A. (Note that this cylinder is two dimensional.)

A robust line corresponds to a cylinder of radius ϵ that encloses at least t points; call such a cylinder a *robust cylinder*. In general, there may be infinitely many solutions to the RSL problem; in the next three paragraphs, we discuss how these solutions are provided as output by the algorithm.

Form of the output. The output is provided in the dual (r, θ) space described earlier, denoted **D**. Recall that each line is mapped to two points: (r, θ), as defined in Section 2, and $(-r, \theta + \pi)$.

The following remark may help to illustrate the role of this mapping in ensuring continuity. Consider a straight line with a fixed orientation that is advanced from the left side of the origin to the right side; as the line is advanced, the mapping (r, θ), $r > 0$, of the line changes by continuously reducing r; when the line crosses the origin, the mapping becomes $(-r, \theta)$ with $r > 0$ still, and then a continuous increase of r follows.

The output is provided as a set of regions, which between them contain the images of the axes of all the robust cylinders. Each axis will appear twice: once for each of its images. Each region is given by providing its boundary, comprising a series of continuous curves, meeting at vertices. Each curve is specified by providing its equation. See Remark 6 below for a further discussion of the output form.

4.1.1 Overview of the Geometric Algorithm

An *image perimeter* will mean the perimeter of a rectangle that contains all input points; it will be convenient to select a large enough *containing rectangle*, so that the distance between each point and an edge of the rectangle is larger than ϵ. It is also convenient to choose the origin to be the lower left hand corner of this rectangle (so that all the image points have positive coordinates). First, we consider an *anchored* instance of the RSL problem, where the cylinder axis is required to pass through a fixed point on the image perimeter, called the *anchor*. To find the robust cylinders, we rotate the axis of the cylinder about the given anchor, keeping track of the points as they enter and leave the rotating cylinder. The rotational order of the point instances *partitions* the angular interval $[0, \Pi]$ into contiguous *angular intervals* (AIs), as follows. Each robust cylinder whose axis lies within the same angular interval contains the same input points. Consider an anticlockwise rotation of a radius ϵ cylinder about the anchor (the rotation corresponds to traversing the angular interval $[0, \Pi]$). For $1 \leq i \leq m$, input point p_i enters the cylinder at some (*entering*) angle e_i and leaves the cylinder at some larger (*leaving*) angle l_i. It is convenient to define the *entering boundary* of the cylinder to be the boundary crossed by p_i at the entering angle e_i; note that the entering boundary is a straight line that maps to a point in the dual space; the *leaving boundary* is defined analogously. We conclude that:

Observation. There are at most $2m + 1$ angular intervals.

Counting the number of points contained in the cylinders associated with each angular interval is easy: associate a count of $+1$ with each entering angle e_i and -1 with each leaving angle l_i; accumulate the counts while traversing the angular interval $[0, \Pi]$). (The above algorithm for the anchored RSL problem can be implemented to run in time $O(m \log m)$; we do not detail it, since we are not interested in this special case.)

For the sequel, we need only remember that the order of the $2m$ (entering and leaving) angles determined the number of points in each cylinder whose axis fell in a given angular interval.

We are ready to consider the RSL problem itself. In a nutshell, our algorithm proceeds as follows: *Maintain the solution for the anchored RSL problem, while advancing the anchor counterclockwise around the image perimeter.*

Specifically, as the anchor is advanced around the perimeter, we show how to maintain the order of the (entering and leaving) angles. This order *partitions* the image perimeter into contiguous *perimeter intervals* (PI), as follows. For each anchor in the same perimeter interval the order of the (entering and leaving) angles is the same. Each point on the perimeter, where the order of an entering/leaving angle and another entering/leaving angle changes is called a *critical anchor.*

Next, we give full characterization of critical anchors. We have two cases.

Case 1. See Figure 2. Consider two input points p_i and p_j whose Euclidean distance is larger than 2ϵ. Suppose that the straight line L_{ij} defined by these two points intersects the image perimeter at points a and b, and that p_i is closer to a than p_j (implying that

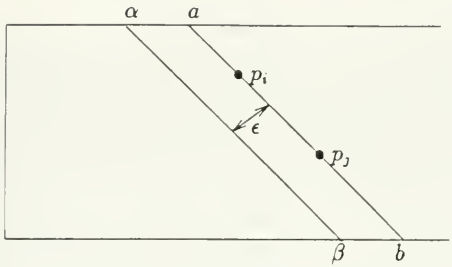


Figure 2: Critical anchors α and β

p_j is closer to b than p_i). The pair of points a and b partitions the containing rectangle (whose perimeter is used for the image perimeter) into two "halves". Advance from a to b counterclockwise to define the first half-perimeter. We focus below on the first half-perimeter. All statements can be adapted to hold for the second half-perimeter, as well. Consider the solution to the anchor RSL problem with respect to anchor a and suppress all entering/leaving angles which relate to points other than p_i and p_j. The relative order of the entering/leaving angles of points p_i and p_j will be e_i, e_j, l_j and l_i. There is exactly one straight line whose distance from L_{ij} is ϵ and which intersects the first half-perimeter. Its two intersection points with this half-perimeter (in counterclockwise order) are denoted α and β.

Lemma 7 *Point α is a critical anchor since the order of e_i and e_j changes there.*

Lemma 8 *Point β is a critical anchor since the order of l_i and l_j changes there.*

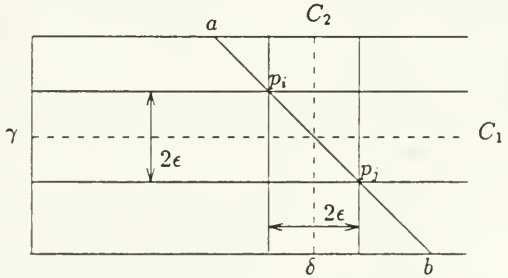


Figure 3: Critical anchors γ and δ

See Figure 3. Consider a radius ϵ cylinder such that point p_i lies on one of its boundaries and point p_j lies on its other boundary. There are exactly two such cylinders, denoted C_1 and C_2. The axis of each of C_1 and C_2 has one intersection point with the first half-perimeter. Denote these two intersection points (in counterclockwise order) by γ and δ.

Lemma 9 *Point γ is a critical anchor since the order of e_i and l_j changes there.*

Lemma 10 *Point δ is a critical anchor since the order of e_i and l_j changes there.*

Case 2. The Euclidean distance between two input points p_i and p_j is at most 2ϵ. The only critical anchors on the first half-perimeter are the points α and β. (Points γ and δ are undefined.)
We conclude:

Theorem 4 *The pair of points p_i and p_j defines at most eight critical anchors. The total number of critical anchors is at most $4m(m-1)$.*

Now, we discuss precisely how the robust cylinders are reported. The goal is to trace the boundaries separating angular intervals representing ϵ cylinders containing t points and ϵ cylinders containing $t-1$ points. We compute the images of these boundaries in the dual space **D**; they comprise the boundaries for the regions of robust axes (or more strictly for the images of these regions in the dual space **D**).

Consider one such boundary for a given perimeter interval. Such a boundary is defined by an image point $p = (a, b)$ on which "just robust" cylinders are incident; specifically, the axis of the corresponding "just robust" cylinder is at distance ϵ from p. The image of the set of lines incident on p (also called the image of p) in the dual space D is the curve

$$r = a\sin\theta - b\cos\theta \tag{4}$$

Two curves might be expected, but in fact points (r, θ) and $(-r, \theta + \pi)$ belong to the same curve.

Next, we determine the equations of the ϵ cylinder axes for which point p is on the cylinder boundary. There are two cases.

Case 1. The axis is above point p on the cylinder boundary. (See Figure 4.) Then the

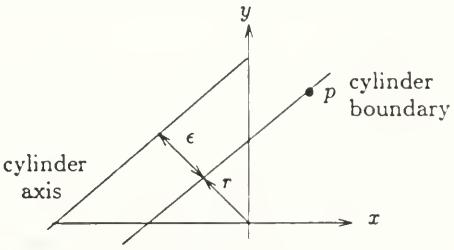


Figure 4: Deriving the equation for a cylinder axis

equation is

$$r - \epsilon = a\sin\theta - b\cos\theta$$

To see this note that if the cylinder boundary has parameters (r', θ), then the axis has parameters $(r' + \epsilon, \theta) = (r, \theta)$; also (r', θ) satisfy equation 4.

Case 2. The axis is below the boundary. Then the equation is

$$r + \epsilon = a \sin\theta - b \cos\theta$$

Point p separates two angular intervals with counts of t and $t-1$, respectively, for some portion of the perimeter delimited by two critical anchors. The ϵ cylinders for which p is on the cylinder boundary are represented in the dual space **D** by a segment of the curves defined in Cases 1 and 2 above (due to the output representation there are actually two segments). The endpoints of the segment are the images in **D** of the axes through the critical anchors. These endpoints are computed as the corresponding critical anchors are traversed.

The following observation clarifies the way in which the dual space is partitioned.

Observation. Suppose the image points are in general position (i.e., no three points are collinear or on a pair of parallel lines distance 2ϵ apart). Then at each endpoint two segments that belong to the same boundary curve meet. There are three ways this can occur. The endpoint can terminate one segment and begin a second one (with respect to the traversal); the endpoint can terminate two segments, or the endpoints can begin two segments. If the general position assumption does not hold then more than two segments may meet at a single endpoint, through collapsing several endpoints into a single endpoint.

The remaining issue is to determine, in **D**, which side of the segment contains the region of robust axes; but this is readily done. If the angular interval with count t begins (resp. ends) when point p is encountered (in the anticlockwise rotation about the anchor) then the side of the segment corresponding to larger (resp. smaller) values of θ is the side containing the robust axes.

The reason for describing the regions of robust axes in the dual space is that a region of lines, in our opinion, is more compact if described as a set of points; if it is intended to present the output on, for instance, a terminal, the presentation in the dual space is much more immediate to the viewer.

This completes the overview. We remark that this approach is based on a solution to the geometric retrieval problem due to Cole and Yap [CY85].

Remark 5. It is tempting to determine that only one of the two images (r, θ) and $(-r, \theta + \pi)$ of a line is required. The difficulty is that we would like to obtain continuous regions of robust axes, which entails avoiding discontinuities at $r = 0$. The problem is that we might not choose consistent mappings for different boundary segments. For a region may have holes; in this situation it is clear that inconsistent mappings for the boundaries of the region and its hole are very undesirable. (With sufficient care, this difficulty may be avoidable: only commit to the mapping at the very end of the computation, at which point consistency of the mappings can be ensured. However, it seems quite intricate to determine exactly what needs to be done; also this has a considerably more adhoc feel to

it. Further, one might want to identify nearby regions, for example; it is not clear that this more intricate approach would lend itself to such a query.)

4.1.2 Implementing the Algorithm

We give two implementations; in this section we describe a method that requires $O(m^2 \log m)$ time. In Section 4.1.4 we give an alternative method that achieves $O(m^2)$ time. In Section 4.1.3, we describe a parallel algorithm that performs $O(m^2 \log m)$ operations in $O(\log m)$ time.

Step 1. Find all critical anchors. This is straightforward; it can be done sequentially in $O(m^2)$ time and in parallel in time $O(1)$ using $O(m^2)$ operations (one processor is assigned to each pair of points).

Step 2. Sort the critical anchors (along the image perimeter). Sequentially, this takes $O(m^2 \log m)$ time; in parallel, using Cole's algorithm [Col88], this takes $O(m^2 \log m)$ operations and $O(\log m)$ time.

Step 3. Advance the anchor around the perimeter in anticlockwise order, determining the boundaries of the regions of robust axes, as follows. A data-structure that supports dictionary operations (i.e., FIND, INSERT and DELETE) is used to maintain the order of the angles relative to the current critical anchor; it is also used to maintain the cumulative counts. (For a sequential implementation, one simple and efficient data structure is the splay tree [ST85].) The order at the next critical anchor will differ only by the interchange of two angles (entering and leaving angles, that is). We need to keep track of the boundary between angular intervals whose cylinders contain t and $t-1$ points, respectively. As, for each leaving/entering angle, we associate the angular interval ahead of it in anticlockwise order, we are seeking entering angles with a count of t and leaving angles with a count of $t-1$. But maintaining the counts is straightforward since the only count that changes is the one for the angular interval between the two angles that are swapped.

The cylinder axis associated with an entering angle ϵ with count t determines a point on the boundary of a region of robust axes in the dual space $\mathbf{D}$ (due to the output representation, there are actually two points). The cylinder axis is determined by some input point p, sustending the angle ϵ; the image of p in $\mathbf{D}$ is a curve; a point on the translation by ϵ of this curve is the image of the cylinder axis. As the anchor is advanced, the image in $\mathbf{D}$ of the cylinder axis comprises a sequence of points which form a segment of the following curve: the translation by ϵ of the image of p. The critical anchor at which the count of entering angle ϵ changes determines, in $\mathbf{D}$, an endpoint of the segment defined by p (more precisely, this endpoint is the image in $\mathbf{D}$ of the cylinder axis for the following cylinder: p is on its entering boundary and its axis passes through the critical anchor). Similar correspondences apply to the leaving angles. So for each entering (resp. leaving) angle with a count of t (resp. $t-1$) we keep track of the corresponding segment in the dual space, and update this information as critical anchors are crossed. At the end of the tour of the image perimeter, complete boundaries for each region of robust axes will have

been specified.

Implementation Comment. Each line that intersects the perimeter of the containing rectangle must intersect it at two points. The line will be reported twice (once through each of these points). To avoid this, we break the processing of the perimeter into four cases, as follows: (Case 1) For the upper edge of the perimeter — all its intersecting lines are considered. (2) For the left edge of the perimeter — only those intersecting lines that do not intersect the upper edge are considered. (3) For the lower edge of the perimeter — only those intersecting lines that do not intersect the upper and left edge are considered. (4) For the right edge of the perimeter — none of its intersecting lines are considered. It should be clear that each such line will be considered exactly once.

4.1.3 A Parallel Algorithm

To obtain m-way parallelism, carry out m instances of the above sequential algorithm, starting at every $4(m-1)$th critical anchor.

Considerably more efficient parallelism can be achieved, but a little less easily. Each segment delimiting a region of robust axes in **D** is defined by its equation and endpoints. A segment equation corresponds to a displacement by ϵ of the mapping in **D** of a point of the image (as described earlier). An endpoint of the segment is obtained from the corresponding critical anchor as follows: let L be the cylinder boundary incident on point p at the critical anchor; the mapping of L in **D** (a point) displaced by ϵ provides the endpoint (the displacement is identical to that used for the mapping of a point in the previous sentence). Thus there are two steps to computing the segments delimiting the regions of robust axes. (1) Determine each delimiting segment. (2) Connect segments having common endpoints. We detail each in turn.

Step 1. For each point p_i, $1 \leq i \leq m$, the segments it determines are computed separately. Let S_i be the set of critical anchors defined by p_i; there are at most $4(m-1)$ such anchors. Let e_i and l_i be the entering and leaving angles sustended by p_i (as the anchor advances around the perimeter). We need to determine the perimeter intervals in which angle e_i (resp. l_i) has an associated count of t (resp. $t-1$), for these intervals determine the "just robust" cylinders which in turn determine the segments delimiting the regions of robust axes in the dual space **D**.

The counts associated with entering angle e_i and leaving angle l_i are constant on the intervals induced by the critical anchors in S_i. So it suffices to compute each set S_i in parallel, and for each set S_i, to compute the counts associated with e_i and l_i. Recall that the count associated with a given entering (or leaving) angle e_i is altered only when a change occurs to the position of e_i in the ordering of entering and leaving angles. These changes occur only at critical anchors. For instance, if in the perimeter interval immediately prior to a critical anchor, in the sorted ordering of the angles, e_i is followed by another entering angle e_j, and if at the critical angle, e_i and e_j interchange positions, then e_i's

count increases by 1 in the next perimeter interval (the interval after the critical anchor in question); similar rules apply in the other cases. Thus, Step 1 is implemented in three substeps, as follows.

Step 1.1. For every i find the count of ϵ_i (and l_i) at some initial anchor location (there will be a single initial anchor location for all ϵ_i and l_i). This is done using the anchored RSL algorithm (see the beginning of section 4.1.1), followed by a prefix sum algorithm [LFS0]. The anchored RSL algorithm takes $O(\log m)$ time and $O(m \log m)$ operations, and the prefix sum algorithm $O(\log m)$ time and $O(m)$ operations.

Step 1.2. For each ϵ_i (and l_i) separately, and in parallel, compute a sorted listing of the changes to the counts associated with ϵ_i. Locations of changes are obtained from the sorted order of the (at most) $2(m-1)$ critical anchors associated with ϵ_i. Using Cole's sorting algorithm this takes $O(\log m)$ time and $O(m \log m)$ operations. Next, a prefix sum algorithm (and the count of ϵ_i at the initial anchor location) is used to determine all the counts for ϵ_i. This prefix sum algorithm runs in $O(\log m)$ time performing $O(m)$ operations. Since Step 1.2 is performed for each ϵ_i and l_i it takes a total of $O(m^2 \log m)$ operations and $O(\log m)$ time.

Step 1.3. For each angle ϵ_i (resp. l_i), relative to S_i, do the following: each time a count of t (resp. $t-1$) is found, determine the corresponding delimiting segment for a region of robust axes – it is a translation by ϵ of a segment of the line which is the image in $\mathbf{D}$ of the point p sustending angle ϵ_i. The endpoints of the segment are critical anchors; specifically they occur at critical anchors where the count associated with ϵ_i changes; for each change in count we need to record the critical anchor and the two points that induce the critical anchor. The computation of the boundaries of the regions of robust axes given the counts takes $O(1)$ time and $O(m)$ operations, or a total of $O(m^2)$ operations and $O(1)$ time for all ϵ_i and l_i. Overall, Step 1 takes $O(\log m)$ time using $O(m^2 \log m)$ operations.

Step 2. Make two copies of each segment, each copy tagged by one endpoint (or the corresponding critical anchor). Sort the copies of the segments using the tag as the key, using Cole's sorting algorithm, for instance. Segments that are to be connected will be adjacent in the sorted order. So Step 2 also takes $O(\log m)$ time using $O(m^2 \log m)$ operations.

Comment. The statement of Step 2 above ignores the (degenerate) case where a critical anchor is induced by more than one pair of points. To cope with this case, we refine the tags associated with each copy of each segment to be pairs, as follows. The first item in a pair is the critical anchor, as before; the second item is the angle in which two points induce the anchor. The previous linear order on tags is now refined to a lexicographic order on these pairs. Sorting is applied again, but this time with respect to the lexicographic order. Another degenerate case arises when more than two segments meet at an endpoint. Then we need to create an adjacency list for each endpoint. Segments that have a common enpdoint will all be contiguous in the sorted order provided by step 2; this set of contiguous segments can provide the adjacency list. (The sorted order is readily partitioned into adjacency lists by an application of a prefix sums algorithm; this takes $O(m^2)$ operations and $O(\log m)$ time.) In addition, in order to facilitate recognition of regions, it is helpful to

have the segments in each adjacency list angularly sorted about their common endpoint. This sort is readily carried out with a further application of Cole's sorting algorithm.

So far we have shown:

Theorem 5 *There is an algorithm to determine the axes of the ϵ robust cylinders for a point set of size m; these axes are reported in the form of regions in the dual plane. Sequentially, the algorithm runs in $O(m^2 \log m)$ time. A parallel implementation performs $O(m^2 \log m)$ operations in $O(\log m)$ time.*

4.1.4 A more efficient algorithm

We now give a more efficient sequential algorithm; it uses the approach of the parallel algorithm above. The key step in the parallel algorithm is the sorting of critical angles associated with each point. We show how to do this in $O(m^2)$ time. It is readily checked that the remainder of the parallel algorithm requires only $O(m^2)$ operations; so it can be carried out sequentially in the same time bound.

We will be using the topological sweep method of Edelsbrunner and Guibas [EGS6]. This procedure, given a collection of n straight lines, determines the arrangement of the lines, and so in particular, for each line, determines the order of its intersections with the other lines; this computation takes time $O(n^2)$. Guibas and Edelsbrunner describe their method for straight lines, but the method can be applied without modification to other classes of lines; of course, it is necessary to verify that the time bounds still hold. We will apply the Guibas and Edelsbrunner method to $2m$ sinusoidal curves, $r = a \sin\theta + b \cos\theta$, with θ in the range $[0, 2\pi]$. We claim that the method still requires $O(m^2)$ time. The key to the complexity bound in [EGS6] is the following lemma. (This explanation is not self contained; the interested reader is referred to [EGS6].) The following definition is helpful: given a region whose boundary consists of line segments, its *size* is the numbers such segments.

Lemma 11 *In an arrangement $A(H)$ of n straight lines, the total size of all the regions whose boundary contains a segment of line l is $O(n)$.*

Proof. (This is a folklore proof; [EGS6] cite two other proofs, namely [CGLS5,EOSS6].) Consider the portions of the lines below line l, and the regions they define bordering l. Name these lines $1, 2, \cdots, n-1$. Traverse the regions bordering l, proceeding along l from one end to the other. As each region is traversed, list the lines encountered in order. The resulting sequence is a list of line names. Since each pair of lines intersects at most once, there is no subsequence of the form *abab*. (The sequence *aba* can occur for the lines are only finite and so a can be nearer to l and then b can become nearer and then end at which point a becomes nearer anew.) It follows that the sequence is a Davenport–Schinzel sequence of type $s = 1$, which has length at most $2n - 1$ [HS86]. The same argument applies to the regions above l. •

Proving the following generalization will ensure that our use of the topological sweep method will take $O(m^2)$ time also.

Lemma 12 *In an arrangement $A(H)$ of n sinusoidal curves, in the range $[0, \pi]$, the total size of all the regions bordering on one of the sinusoidal curves l (having as a side part of l) is $O(n)$.*

Proof. The proof of Lemma 11 follows from the fact that each pair of lines intersects at most once. This is also true for sinusoidal curves in the range $[0, \pi]$. So the proof of Lemma 11 can be applied. •

It remains to describe how to sort the critical anchors for some point p. For each point p we create two curves in the dual space D. The first curve represents the axes of cylinders for which p is on the entering boundary and the second curve the axes of cylinders for which p is on the leaving boundary. If $p = (a, b)$, these are the curves $r \pm \epsilon = a \sin\theta - b \cos\theta$. Each critical anchor is associated with one intersection and all intersections correspond to critical anchors. Further, for each curve, the ordering of intersection points along the curve is exactly the rotational order of the corresponding critical anchors. The topological sweep method is used to obtain the ordering of intersection points along each curve. This yields, for each point p, two sorted sets of critical anchors; merging these two sets provides the sorted ordering of p's critical anchors. Clearly, this takes $O(m^2)$ for all m image points.

We have shown:

Theorem 6 *There is an algorithm to determine the axes of the ϵ robust cylinders for a point set of size m; these axes are reported in the form of regions in the dual plane. The algorithm runs in $O(m^2)$ time.*

An interesting question is whether a parallel version of this implementation can be found. Indeed, there is a parallel algorithm for computing the arrangement of a set of n straight lines in $O(\log n)$ time using $O(n^2)$ operations [Good90]. However, it is not clear whether this algorithm extends to other classes of curves, such as sinusoidal curves.

Remark 6. Clearly, the output can be visualized in the dual space. In addition, it may be desirable to provide a list of the robust lines in a discretized version of the dual space. That is, in the dual space a grid with separation δ, for some $\delta > 0$ is used; each grid point falling within one of the regions of robust axes is reported (or perhaps different separations δ_1 and δ_2 in the θ and r directions are more appropriate).

4.2 Applying Sampling

As in Section 3, we assume that $t \geq 2 \ln m + 2$. Again, we select each point in M with probability $p \geq (2 \ln m + 2)/t$ into a set S. The following lemma provides some observations. Description of an algorithm follows.

Lemma 13 *Let A denote a line. Suppose some r points in S are ϵ-near to A. For any β and p, $0 \leq \beta \leq 1$ and $0 \leq p \leq 1$:*

1. *If A is robust, $r \geq (1-\beta)tp$ with probability at least $1 - \exp(-\beta^2 tp/2)$.*
2. *If $r \geq (1+\beta)tp$, then A is robust with probability at least $1 - \exp(-\beta^2 tp/3)$.*
3. *If $r \geq (1-\beta)tp$, then at least $\frac{1-\beta}{1+\beta}t$ points in M are ϵ-near to A with probability at least $1 - \exp(-\beta^2 \frac{1-\beta}{1+\beta} tp/3)$.*

An algorithm that uses sampling: We apply the geometric algorithm already described in this section to the sample S, using $(1-\beta)tp$ as the threshold.

In order to apply Lemma 13 to the algorithm we need to bound the number of lines A that need to be considered. We show a bound of $8|M|^2$ lines.

Definition A *cylinder class* is a collection of cylinders all of which have the same point(s) on their boundaries and the same points in their interior.

Lemma 14 *There are at most $8|M|^2$ cylinder classes.*

Proof. By Theorem 3 there are at most $2|M|^2$ cylinders with two (or more) points on their boundaries (for there are exactly $4|M|(|M|-1)$ critical anchors each defining such a cylinder; but each cylinder is counted twice). Next we consider cylinder classes that have only one point on their boundary: for each such cylinder, rotate it anticlockwise about its boundary point until a second point is on its boundary. Since there are at most $2|M|^2$ cylinders with two points on boundary, there are at most $4|M|^2$ classes of cylinders with at least one point on their boundary. Finally, we consider cylinder classes that have no points on their boundary; for each such cylinder, translate it upwards until a first point is on its boundary. Since there are at most $4|M|^2$ cylinders with at least point on their boundary, there are at most $8|M|^2$ classes of cylinders. •

Now, we apply Lemma 13(3) to each class of cylinders and then use Proposition 2 to obtain: with probability at least $1 - 8|M|^2 \exp(-\beta^2 \frac{1-\beta}{1+\beta} tp/3)$, the regions reported by the algorithm contain only axes of cylinders holding at least $(1 - \frac{1-\beta}{1+\beta})t$ points. Of course, as noted earlier, in Remark 1, this is likely to be a rather small probability. Still, as before, we expect to report most of the regions of robust lines, and possibly some other regions in addition. Again, a postprocessing phase is called for to filter the results computed here.

The sampling reduces the runtime of the previous algorithm to $O(|M| + |S|^2 \log |S|)$, but at the loss of some certainty regarding finding the complete regions of robust cylinders.

4.3 The Segment-RSL Problem

In turn, we describe a sequential and a parallel algorithm. Both perform $O(|S|^2 \log |S|)$ operations, though in practice we suspect they would be considerably more efficient; this is discussed further later.

4.3.1 The Sequential Algorithm

We carry out essentially the geometric algorithm described earlier. First we need to define a few terms. For a given anchor, an angle α is *legal* for segment σ if the line A through the anchor, sustending angle α, has the property that σ is ϵ-near to A. For a given anchor, a segment is *legal* if there is some legal angle for that segment.

The new algorithm comprises essentially Step 3 of the previous geometric algorithm. The current anchor is swept around the perimeter of the image. For the current anchor, we keep a balanced tree containing an entering and a leaving angle for each segment σ; the entering angle is the smallest angle for which σ is legal, and the leaving angle is the largest such angle. The angles are stored in rotational order. As before, changes to the ordering of the angles occur at critical anchors. In order to determine the next critical anchor, we keep a heap of candidate future critical anchors. This heap initially stores the following critical anchors: first, those critical anchors defined for each segment σ as follows: the anchors at which σ becomes and ceases to be legal. Second, each critical anchor defined by neighboring angles in the balanced tree (such a critical anchor occurs when two entering and or leaving angles interchange their relative order). In the next paragraph, we discuss how the heap and balanced tree are updated as the current anchor is advanced. Two paragraphs hence, we explain how the balanced tree is used to determine the boundaries of the regions of robust axes.

The processing of Step 3 is straightforward. For some initial anchor, the entries in the balanced tree are determined and the associated heap is computed. Next, repeatedly, the minimum item is removed from the heap (the initial anchor is assumed to be the smallest anchor); the item removed from the heap is the next critical anchor to be processed. If it represents a newly legal segment σ, the entering and leaving angles associated with σ are added to the balanced tree, and two new values are added to the heap (i.e., the critical anchors determined by the new entering and leaving angles and their neighbours in the balanced tree); finally, one item is deleted from the heap – the critical anchor corresponding to the two no longer adjacent angles. A segment which ceases to be legal is processed similarly. The other possibility is that the critical anchor corresponds to an interchange in the order of two angles; then their order in the balanced tree is updated and appropriate insertions and deletions are made to the heap (two deletions and two insertions).

In order to determine the regions of robust axes, we need to keep track of the limits of the angular regions which contain segments of total length at least l. In order to do this, we keep a cummulative sum as before. But now for the entering angle corresponding to a line of length l' we have an additive term of $+l'$, while for the leaving angle we add a term of $-l'$. The boundary occurs whenever the sum goes from $\geq l$ to $< l$, or from $< l$ to $\geq l$. The image of this boundary in the dual space D is computed and reported as before.

It remains to discuss how to compute the leaving and entering angles for a segment

σ. A little thought shows that the entering angle is determined by the second endpoint of σ to cross the entering boundary of the ϵ-cylinder (as the cylinder is rotated about the current anchor). Similarly the leaving angle is determined by the first endpoint of σ to cross the leaving boundary of the ϵ-cylinder. Clearly, if the leaving angle is smaller than the entering angle, the segment is not legal. In addition, two angles become equal when the line through the associated endpoints determines the current anchor; this is computed as before.

This procedure requires $O(\log |S|)$ time to process each critical anchor. In addition, the time required to initialize the heap and the balanced tree is $O(|S| \log |S|)$. As before, there are $O(|S|^2)$ critical anchors. We have shown:

Theorem 7 *There is an algorithm to determine the axes of the ϵ robust cylinders for a set S of segments. These axes are reported in the form of regions in the dual plane. The algorithm runs in $O(|S|^2 \log |S|)$ time.*

In fact, a critical anchor occurs for a pair of segments only if there is a line A such that both segments are ϵ-near to A. It is plausible to assume that in practice the size of this set is much smaller that $|S|$. So we define the *overlap count* of segment σ to be the number of segments $\sigma' \in S$ such that there is a line A to which both σ and σ' are ϵ-near. Suppose the overlap count for each segment in S is bounded by r. Then we have shown:

Corollary 15 *There is an algorithm to determine the axes of the ϵ robust cylinders for a set S of sgements. These axes are reported in the form of regions in the dual plane. If the overlap count for each segment in S is bounded by r, the algorithm runs in $O(r \cdot |S| \log |S|)$ time.*

Sampling could be applied here too, with a segment being selected with a probability proportional to its length. But unless the number of segments defining the typical robust line is large, sampling will not be particularly helpful. We supsect that the situations in which Corollary 15 is useful will prove more typical.

Again, we could describe an algorithm with an $O|S|^2)$ performance. However, as it is not clear that a complexity of the type stated in Corollary 15 can be achieved, we are not exploring this further here.

Remark 7. We comment here on the definition of the segment-RSL problem. First, in practice it may be appropriate to ignore short segments for it is not clear that they convey useful information concerning a robust line. Thus, for example, one might exclude all lines of length less that 10ϵ. Second, one might want to require that in addition to being ϵ-near to a robust line, a segment should have a nearby orientation also. So we could put a bound of α on the legal range of orientations. This additional restriction is easily implemented in our algorithm; it simply requires a modification to the entering and leaving angles: now they are restricted to deviate no more than α from the orientation of the segment. Third, instead of using the length l of a segment to measure its contribution to a robust

line, it might be desirable to use the projected length. But again, this is readily carried out: the projected length will be a term of the form $a \sin\theta + b\cos\theta$, where θ denotes the orientation of the candidate robust line and a and b are constants determined by the length and orientation of the segment. If the first rule is implemented, the second rule will only have a modest effect, for the difference in lengths will be less than 1%.

4.3.2 The Parallel Algorithm

.

The parallel algorithm is very similar to the previous parallel algorithm. For each segment, we compute the portions of the boundaries of the regions of robust axes it determines. Specifically, for each segment σ, we compute a sequence of counts associated with its entering and leaving angles, respectively. This is computed for the range of anchors for which the segment is legal. We explain this in more detail for the entering angle. For each segment σ' we determine the range of anchors such that its entering and leaving angles straddle the entering angle of σ (this is determined by the critical angles induced by σ and σ'). With this range we associate an increment l', equal to the length of σ'. In order to sum these increments, we associate an increment of $+l'$ with the start of the range and an increment of $-l'$ with the end of the range. We compute the prefix sums of the increments. A change, in the prefix sum, as before, from $< l$ to $\geq l$, or conversely, corresponds to the segment σ determining a portion of the boundary of a region of robust axes. This is handled as in the previous parallel algorithm. As in the previous parallel algorithm, we use a sorting algorithm and a prefix sums algorithm, and for each segment they are being applied to $O(|S|)$ items. We have shown:

Theorem 8 *There is a parallel algorithm to determine the axes of the ϵ robust cylinders for a set S of segments. These axes are reported in the form of regions in the dual plane. The algorithm performs $O(|S|^2 \log|S|)$ operations in $O(\log|S|)$ time.*

We have not been able to find an algorithm which admits a complexity similar to that specified by Corollary 15. We leave this as an open problem.

4.4 Further Directions

One natural question is how to choose ϵ. One approach that suggests itself is to use an adaptive strategy: For instance, to repeatedly double the value of ϵ until the regions of robust axes cease to grow significantly (obviously these regions will grow slightly as ϵ increases).

A possible extension of this approach is to adapt ϵ to each region, for one can imagine that there might be different densities of points or segments in different portions of the image.

Acknowledgments. We had helful discussions with Yossi Matias and Ehud Rivlin. Azriel Rosenfeld drew our attention to the problem of reconstructing lines from segments. Sanjeev Khudanpur ran the experiment described in Section 3. Last, but not least, Larry Davis, Robert Hummel, David Mount and Haim Wolfson gave us comments on the manuscript. All this help is gratefully acknowledged.

References

[AV79] D. Angluin and L.G. Valiant. Fast probabilistic algorithms for Hamiltonian circuits and matchings. *J. Comput. and Sys. Sci*, 18:155-193, 1979.

[ACG89] M.J. Atallah, R. Cole, M.T. Goodrich. Cascading divide and conquer: a technique for designing parallel algorithms. In *SIAM J. Comput.*, 3:499-532, 1989.

[AG86] M.J. Atallah and M.T. Goodrich. Efficient plane sweeping in parallel. In *Proc. 2nd ACM Symp. on Computational Geometry*, 216-225, 1986.

[BB82] D.H. Ballard and C.M. Brown. Computer Vision. Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1982.

[BS90] J.R. Bergen and H. Shvaytser. A probabistic algorithm for computing Hough Transform. Preprint, 1990.

[Br83] C.M. Brown. Inherent bias and noise in the Hough Transform. *IEEE Trans. of Pattern Recognition and Machine Intelligence*, PAMI-5,5:493-505, 1983.

[CGL85] B.M. Chazelle, L.J. Guibas, D.T. Lee. The power of geometric duality. *BIT* 25:76–90, 1985.

[Col88] R. Cole. Parallel merge sort. *SIAM Journal on Computing*, 770-785, 1988.

[CY85] R. Cole and C.K. Yap. Geometric retrieval problems. *Information and Computation*, 63(1985), 39-57.

[DKM+88] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Ronnert and R.E. Tarjan. Dynamic perfect hashing: upper and lower bounds. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 524-531, 1988.

[DL76] D.P. Dobkin and R. Lipton. Multidimensional searching problems. In *SIAM J. Comput.*, 5:181-186, 1976.

[EG86] H. Edelsbrunner and L. Guibas. Topologically sweeping an arrangement. *Proc. 18th Annual ACM Symp. on Theory of Computing*, 389-403.

[EOS86] H. Edelsbrunner, J. O'Rourke, R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. on Computing*, 15:341–363, 1986.

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Comm. of the Association for Computing Machinery*, 24,6:381-395, 1981.

[FF81] M.A. Fischler and O. Firschein. Parallel guessing: a strategy for high-speed computation. *Pattern Recognition*, 20:257-263, 1987.

[FKS84] M.L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. of the Association for Computing Machinery*. 31:538–544, 1984.

[GH90] W.E.L. Grimson and D.P. Huttenlocher. On the sensitivity of the Hough Transform for object recognition. *IEEE Trans. of Pattern Recognition and Machine Intelligence*, PAMI-12,3:255-274,1990.

[Good90] M. Goodrich. Constructing arrangements optimally in parallel. Manuscript.

[HR89] T. Hagerup and C. Rub. Optimal Merging and Sorting on the EREW PRAM. Technical Report, Universität des Saarlandes, 1989.

[HS86] S. Hart and M. Sharir. Nonlinearity of Davenport–Schinzel sequences of a generalized path compression scheme. *Combinatorica* 6:151–177, 1986.

[IK88] J. Illingworth and J. Kittler. A survey of the Hough transform. *Computer Vision, Graphics, and Image Processing*, 44:87–116, 1988.

[LF80] R.E. Ladner and M.J. Fischer. Parallel prefix computation. *J. Association for Computing Machinery*, 27:831-838, 1980.

[MV90] Y. Matias and U. Vishkin. On parallel hashing and integer sorting. UMIACS-TR-90/13, Inst. for Advanced Computer Studies, University of Maryland, January 1990. Also, On integer sorting and parallel hashing. Proc. 17th ICALP, Lecture Notes in Computer Science, 443, Springer-Verlag, 1990, 729-743; to appear, *J. Algorithms*.

[Me84] K. Mehlhorn. *Data structures and algorithms 3: Multi-dimensional searching and computational geometry*. Springer Verlag, 1984.

[Ra76] M.O. Rabin. Probabilistic algorithms. In *Algorithms and Complexity*, J.F. Traub, Editor, Academic Press, 21-39, 1976.

[RR89] S. Rajasekaran and J.H. Reif. Optimal and sublogarithmic time randomized parallel sorting algorithms. *SIAM J. Comput.*, 18:594-607, 1989.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. In *C.ACM*, 29:669-679, 1986.

[ST85] D.D. Sleator, R.E. Tarjan. Self-adjusting binary search trees. *J.ACM*, 652-686, 1985.

[XOK90] L. Xu, E. Oja and P. Kultanen. A new randomized curve detection method: Randomized Hough Transform (RHT). *Pattern Recognition Letters*, 331-338, 1990.